APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: QUEUE ARRAY IN NETWORK DEVICES

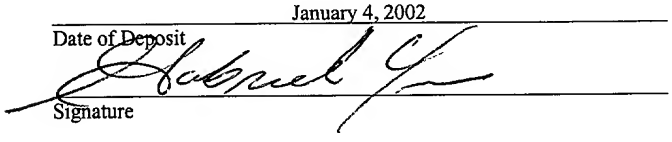
APPLICANT: GILBERT WOLRICH, DEBRA BERNSTEIN AND MARK B. ROSENBLUTH

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV 044489962US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

January 4, 2002
Date of Deposit


Signature

Gabe Lewis
Typed or Printed Name of Person Signing Certificate

Queue Arrays in Network Devices

BACKGROUND

This invention relates to utilizing queue arrays in network devices.

Some network devices such as routers and switches have line speeds that can be faster than 10 Gigabits. For maximum efficiency the network devices' processors should be able to process data packets, including storing them to and retrieving them from memory at a rate at least equal to the line rate. However, current network devices may lack the necessary bandwidth between their processors and memory to process data packets at the devices' line speeds.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a network system.

FIG. 2 is a block diagram of a network device.

FIG. 3 shows a queue and queue descriptor.

FIG. 4 is a block diagram of a network processor's cache.

FIG. 5 is a flow chart illustrating an enqueue operation.

FIG. 6 is a flow chart illustrating a dequeue operation.

FIG. 7 is a flow chart illustrating a fetch operation.

DETAILED DESCRIPTION

As shown in FIG. 1, a network system 2 for processing data packets includes sources of data packets 4 coupled to a network device 6 and destinations for data packets 8 coupled to the network device 6. The network device 6 includes a processor 10 with memory data structures configured to receive, store and forward the data packets to a specified destination. The network device 6 can include a network switch, a network router or other network device. The source of data packets 4 can include other network devices connected over a communications path operating at high data packet transfer line speeds. Examples of such communications paths include an optical carrier (OC)-192 line, and a 10-Gigabit line. Likewise, the destination 8 of data packets also can include other network devices as well as a similar network connection.

As shown in FIG. 2 the network device 6 includes memory 14 coupled to the processor 10. The memory 14 stores output queues 18 and their corresponding queue descriptors 20. Upon receiving a data packet from a source 4 (FIG. 1), the processor 10 performs enqueue and dequeue operations to process the packet. An enqueue operation adds information that has arrived in a data packet, which previously was stored in memory 14, to one of the output queues 18 and

updates its corresponding queue descriptor 20. A dequeue operation removes information from one of the output queues 18 and updates the corresponding queue descriptor 20, thereby allowing the network device 6 to transmit the information to an appropriate destination 8.

An example of an output queue 18 and its corresponding queue descriptor is shown in FIG. 3. The output queue 18 includes a linked list of elements 22, each of which contains a pointer 24 to the next element 22 in the output queue 18. A function of the address of each element 22 implicitly maps to the information 26 stored in the memory 14 that the element 22 represents. For example, the first element 22a of output queue 18 shown in FIG. 3 is located at address A. The location in memory of the information 26a that element 22a represents is implicit from the element's address A, illustrated by dashed arrow 27a. Element 22a contains the address B, which serves as a pointer 24 to the next element 22b in the output queue 18, located at address B.

The queue descriptor 20 includes a head pointer 28, a tail pointer 30 and a count 32. The head pointer 28 points to the first element 22 of the output queue 18, and the tail pointer 30 points to the last element 22 of the output queue 18. The count 32 identifies the number (N) of elements 22 in the output queue 18.

Enqueue and dequeue operations for a large number of output queues 18 in memory 14 at high bandwidth line rates can be accomplished by storing some of the queue descriptors 20 in a cache 12 at the processor's 10 memory controller 16 (FIG. 2). Commands to perform enqueue or dequeue operations reference queue descriptors 20 presently stored in the cache 12. When an enqueue or a dequeue operation is required with respect to a queue descriptor 20 that is not presently in the cache 12, the processor 10 issues commands to the memory controller 16 to remove a queue descriptor 20 from the cache 12 to the memory 14 and to fetch a new queue descriptor 20 from memory 14 for storage in the cache 12. In this manner, modifications to a queue descriptor 20 made by enqueue and dequeue operations occur in the cache 12 and are copied to the corresponding queue descriptor 20 in memory 14 upon removal of that queue descriptor 20 from the cache 12.

In order to reduce the read and write operations between the cache 12 and the memory 14, it is possible to fetch and return only those parts of the queue descriptor 20 necessary for the enqueue or dequeue operations.

FIG. 4 illustrates the contents of the cache 12 used to accomplish this function according to one particular implementation. In addition to a number of queue descriptors 20 corresponding to some of the queue descriptors stored in the memory 14, the cache 12 designates

a head pointer valid bit 34 and a tail pointer valid bit 36 for each queue descriptor 20 it stores. The valid bits are set when the pointers to which they correspond are modified while stored in the cache 12. The cache 12 also tracks the frequency with which queue descriptors have been used. When a command requires the removal of a queue descriptor, the least-recently-used ("LRU") queue descriptor 20 is returned to memory 14.

As illustrated by FIG. 5, when performing an enqueue operation, the processor 10 checks 40 if a queue descriptor 20 for the particular queue 18 to which the information will be attached is in the cache 12. If it is not, the processor 10 removes 42 the least-recently-used queue descriptor 20 from the cache 12 to make room for the requested queue descriptor. The tail pointer 30 and count 32 of the requested queue descriptor 20 are fetched 44 from memory 14 and stored in the cache 12, and the tail pointer valid bit (Vbit) 36 is set 46. The processor 10 then proceeds with the enqueue operation at block 60.

If (at block 40) the queue descriptor 20 for the particular requested queue 18 is already in the cache 12, the processor 10 checks 48 whether the tail pointer valid bit 36 has been set. If it has not been set, the tail pointer 30 is fetched 50 from memory 14 and stored in the queue descriptor 20 in the cache 12, and the tail pointer

valid bit 36 is set 46. The processor 10 then proceeds with the enqueue operation at block 60. If (at block 48) the tail pointer valid bit 36 has been set, the processor proceeds directly to the enqueue operation at block 60.

In block 60, the processor 10 determines whether the output queue 18 is empty by checking if the count 32 is set to zero. If the count 32 is set to zero, the output queue 18 is empty (it has no elements 22 in it). The address of the new element 22 which implicitly maps to the new information 26, the information 26 being already in the memory 14, is written 62 in both the head pointer 28 and tail pointer 30 in the cache 12 as the new (and only) element 22 in the output queue 18. The count 32 is set 64 to equal one and the head pointer valid bit is set 66.

If (at block 60) the count 32 is not set to zero and the output queue 18 is, therefore, not empty, the processor links 68 the address of the new information's 26 element 22 to the pointer 24 of the last element 22. Thus the pointer 24 of the last element 22 in the queue 18 points to a new element 22 representing the new information 26. The processor 10 writes 70 the address of this new element 22 to the tail pointer 30 of the queue descriptor 20 in the cache 12. The processor 10 increments 72 the count by one and the Enqueue operation is then complete.

FIG. 6 illustrates a dequeue operation. The processor 10 checks 80 whether the queue descriptor 20 for the particular output queue to be used in the dequeue operation is presently in the cache 12. If it is not, the processor 10 removes 81 a queue descriptor from the cache 12 to make room for the requested queue descriptor 20. The processor 10 then fetches 82 the head pointer 28 and count 32 of the requested queue descriptor 20 from memory 14, stores them in the cache 12 and sets 84 the head pointer valid bit (Vbit). The processor 10 proceeds with the dequeue operation at block 90.

If (at block 80) the queue descriptor 20 for the particular output queue 18 requested is already in the cache 12, the processor checks 86 whether the head pointer valid bit 34 has been set. If it has not been set, the head pointer 28 is fetched 88 and the processor 10 proceeds with the dequeue operation at block 90. If the head pointer valid bit 34 has been set, the processor 10 proceeds directly to the dequeue operation at block 90.

In block 90, the head pointer 28 is read to identify the location in memory 14 of the first element 22 in the output queue 18. The information implicitly mapped by the element's 22 address is to be provided as output. That element 22 is also read to obtain the address of the next element 22 in the output queue 18. The address of the next

element 22 is written into the head pointer 28, and the count 32 is decremented.

The head pointer 28 need not be fetched during an enqueue operation, thereby saving read bandwidth between the processor 10 and memory 14. Similarly, a tail pointer 30 need not be fetched from memory 14 during a dequeue operation. When a queue descriptor 20 is removed 42, 81 from the cache 12, the processor 10 checks the valid bits 34, 36. If there were no modifications to the tail pointer 30 (for example, when only dequeue operations were performed on the queue), the tail pointer valid bit 36 remains unset. This indicates that write bandwidth can be saved by writing back to memory 14 only the count 32 and head pointer 28. If there were no modifications to the head pointer 28 (for example, when only enqueue operations to a non-empty output queue 18 were performed), the head pointer valid bit 34 remains unset. This indicates that only the count 32 and tail pointer 30 need to be written back to the queue descriptor 20 in memory 14, thus saving write bandwidth.

In some implementations, when a particular queue descriptor 20 is used in the cache 12 for a second time, a "fetch other" operation is executed before the enqueue or dequeue operation. As shown by FIG. 7, one implementation of the "fetch other" operation 94 causes the processor 10 to determine 94 whether the head pointer valid bit 34 has been

set and to fetch 95 the head pointer 28 from memory 14 if it has not. If the head valid bit 34 has been set, the processor 10 checks 96 whether the tail valid bit 36 has been set and, if it has not, fetches 97 the tail pointer 30. At completion of the "fetch other" operation, both the head valid bit 34 and the tail valid bit 36 are set 98.

The use of both pointers is needed only if the second enqueue or dequeue operation with respect to the queue descriptor 20 is not the same as the first such operation. However excess bandwidth to support this possibly superfluous fetch and return of queue descriptor 20 parts 28, 30 can be available when the queue descriptor is used by operations more than once while stored in the cache 12.

Various features of the system can be implemented in hardware, software or a combination of hardware and software. For example, some aspects of the system can be implemented in computer programs executing on programmable computers. Each program can be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. Furthermore, each such computer program can be stored on a storage medium, such as read only memory (ROM) readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage medium is read by the computer to perform the functions described above.

Other implementations are within the scope of the following claims.